Manipulating Correspondences for Stepwise Camera Gluing

A[E]4M33TDV—3D compute vision: labs.

Martin Matoušek

November, 2010*

Lecture Prerequisities: Stepwise Gluing.

1 Introduction

The Stepwise Gluing algorithm constructs cluster of cameras and cloud of 3D scene points. It works with pointwise correspondences. Tentative image-to-image correspondences, that relate interest points between pairs of images (cameras), are the input of the algorithm. These have been obtained e.g. by WBS matcher. Additionally, scene-to-image correspondences, tentative as well as confirmed, are created and manipulated. These correspondences relate 3D scene points in the cluster and interest points in the images.

While attaching the camera to the cluster, the image-to-image correspondences must be correctly propagated between camera pairs to form scene-to-image correspondences. Here, a method for correspondence manipulation is proposed, together with data structures for representing the correspondences. This correspondence manipulation method is implemented as the Matlab mini-toolbox `corresp`, which can be used by students.

Note, that the correspondence manipulation is driven by the Stepwise Gluing. The information which tentative correspondences are confirmed, which points are created, etc, must be given (it comes from appropriate geometric algorithms, e.g. P3P with RANSAC).

2 Correspondence Manipulation

2.1 Data Structures

Identifiers. Image points as well as scene points have identifiers (IDs). These identifiers can e.g. serve as indices to tables with point coordinates. The identifiers must be given by user. The scene points identifiers must be unique, the image point identifiers must be unique in the scope of a single image. Cameras have also identifiers, these must be integers from 1 to n (number of cameras), later denoted as $i, i_1, i_2, \ldots$

Image-to-Image Correspondences. For every image pair (i_1, i_2), the image-to-image correspondences are represented as a matching table, where each row contains a pair of image point IDs to i_1 and i_2. Internally, the tables are stored in the cell matrix `m`, such that the `m{i1,i2}` is the matching table for the pair (i_1, i_2), assuming that $i_1 < i_2$. So the cell matrix `m` has diagonal and under-diagonal entries empty. The case when $i_1 > i_2$ is internally treated by appropriate swapping of indices and columns, e.g. by using `m{i2,i1}(:,[2 1])`. So the user of the mini-toolbox need not care about.

*Last revision: December 2, 2010

Scene-to-Image Correspondences. For each camera i, the correspondences between scene points and image points are stored in a table, where each row contains a pair of scene point ID and image point ID. These are stored in the cell matrix `Xu`, where `Xu{i}` belongs to the camera i.

2.2 Algorithm

The correspondence manipulation can be summarised in the following steps.

1. **Data Setup**—initialisation of internal correspondence structures.
   Functions: `corresp_init`, `corresp_add_pair`.
2. **Initialisation of Camera Cluster and Point Cloud**. The first two selected cameras establish the camera cluster and inlier image-to-image correspondences create the 3D point cloud.
   Functions: `corresp_start`.
3. **Propagation of Tentative Scene-to-Image Correspondences (1)** of the newly created scene points. Done automatically.
4. **Attaching New Camera** into the cluster.
   Functions: `corresp_join_camera`.
5. **Propagation of Tentative Scene-to-Image Correspondences (2)** of inliers.
   Functions: done automatically.
6. **Reconstructing New Points** from remaining image-to-image correspondences between the attached camera and the other cameras in the cluster.
   Functions: `corresp_new_x`.
7. **Propagation of Tentative Scene-to-Image Correspondences (3)** of new points.
   Functions: done automatically.
8. **Verification of Tentative Scene-to-Image Correspondences in the cluster** that have emerged during previous propagations.
   Functions: `corresp_verify_x`.
9. **Propagation of Tentative Scene-to-Image Correspondences (4)** of the newly confirmed ones.
   Functions: done automatically.
10. **Finalize New Camera**—ensures data consistency.
    Functions: `corresp_finalize_camera.`
11. **Repeat Step 4** while there exists a camera outside the cluster with sufficient number of tentative scene-to-image correspondences.

Particular steps now described in detail, accompanied with a demonstration run on a simple example.

2.2.1 Data Setup

The data structure must be first initialised with a number of cameras.

```
corresp = corresp_init(5);
```

Then all tentative image-to-image correspondences must be imported.

```
corresp = corresp_add_pair( corresp, 1, 2, [ 4 1; 5 1; 3 3; 2 2] );
corresp = corresp_add_pair( corresp, 2, 3, [ 3 3; 3 4] );
corresp = corresp_add_pair( corresp, 1, 3, [ 1 1; 4 2; 6 1; 7 2] );
corresp = corresp_add_pair( corresp, 1, 4, [ 4 3; 4 4] );
corresp = corresp_add_pair( corresp, 2, 4, [ 1 3; 3 4;4 5] );
corresp = corresp_add_pair( corresp, 3, 4, [ 1 3; 5 5 ] );
corresp = corresp_add_pair( corresp, 1, 5, [ 1 1 ] );
```

The situation is depicted on the following graph of correspondences between images, also the content of correspondence tables `m{i,j}` is shown (Fig. 1). Note that the scene-to-image correspondence tables `Xu{i}` are empty at the beginning.

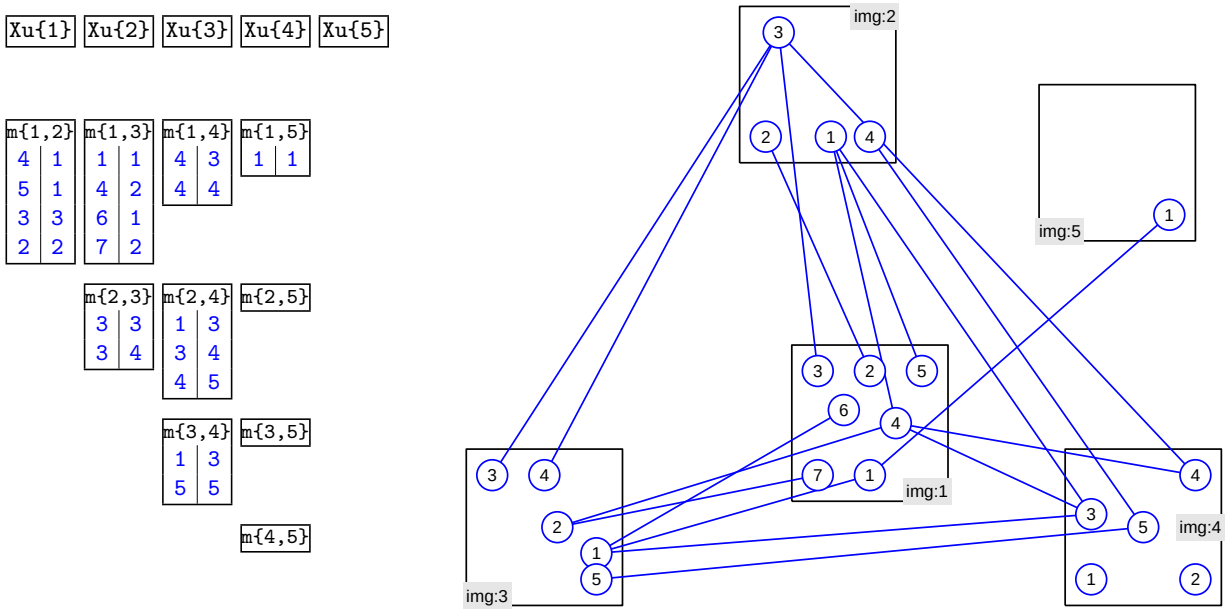

Fig. 1: Starting State.

2.2.2 Initialisation of Camera Cluster and Point Cloud

Two cameras i_1 and i_2 are selected to be the seed of the cluster. Calibrated epipolar geometry between the cameras is estimated and inlier image-to-image correspondences are found. The cameras $\mathbf{P}_{i_1}$, $\mathbf{P}_{i_2}$ and set of 3D scene points are reconstructed.

```
i1 = 1; i2 = 2; % need not be ordered, the function takes care itself
m12 = corresp_get_m( corresp, i1, i2 ); % get image-to-image correspondences

Estimateepipolar geometry using correspondences m12. Obtain set of inliers, reconstruct cameras and
scene points.

inl = [1 3]; % indices to m12 – obtained inliers
xid = ['A' 'B']; % IDs of the reconstructed scene points
corresp = corresp_start( corresp, i1, i2, inl, xid );
```

Now the cluster of cameras consists of these two cameras and the point cloud of these points. Inlier image-to-image correspondences are transferred into confirmed scene-to-image correspondences in i_1 and i_2 and all image-to-image correspondences between i_1 and i_2 are removed. Note that the confirmed correspondences can exist only in the cameras belonging to the cluster. This will be more clear later.

The situation is depicted in the following two figures. The first one (Fig. 2) emphasises image-to-image inliers and outliers, new scene points and new scene-to-image correspondences. The second one (Fig. 3) shows the final state.

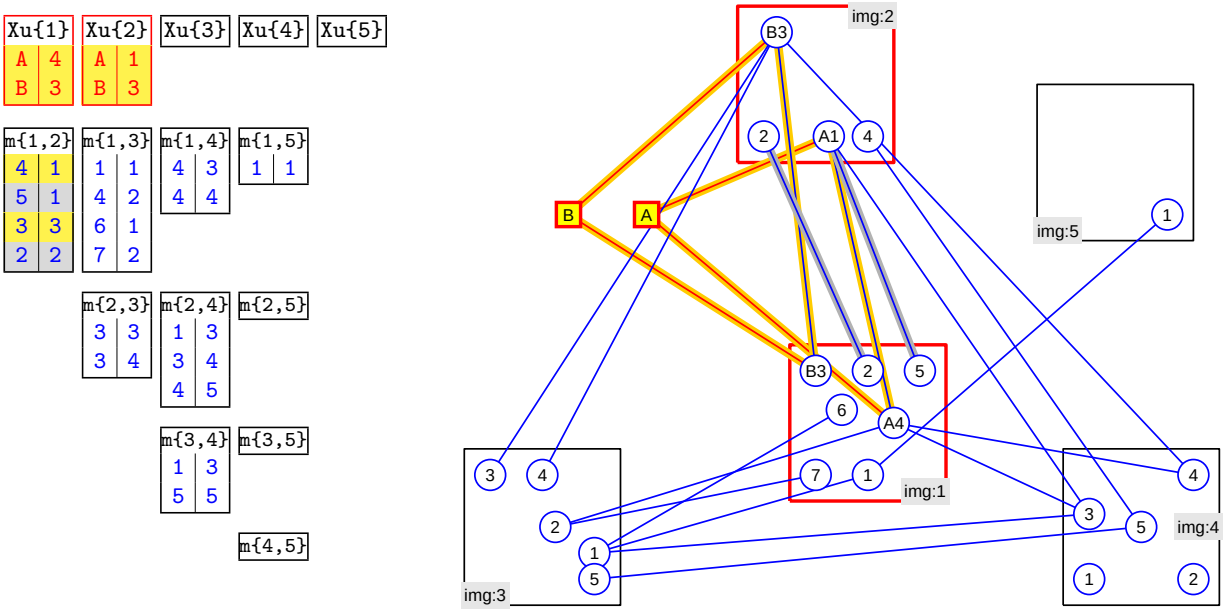

Fig. 2: The camera cluster and the point cloud initialisation in the progress. Emphasized: image-to-image inliers (yellow), outliers (gray), and new points with confirmed scene-to-image correspondences (yellow).

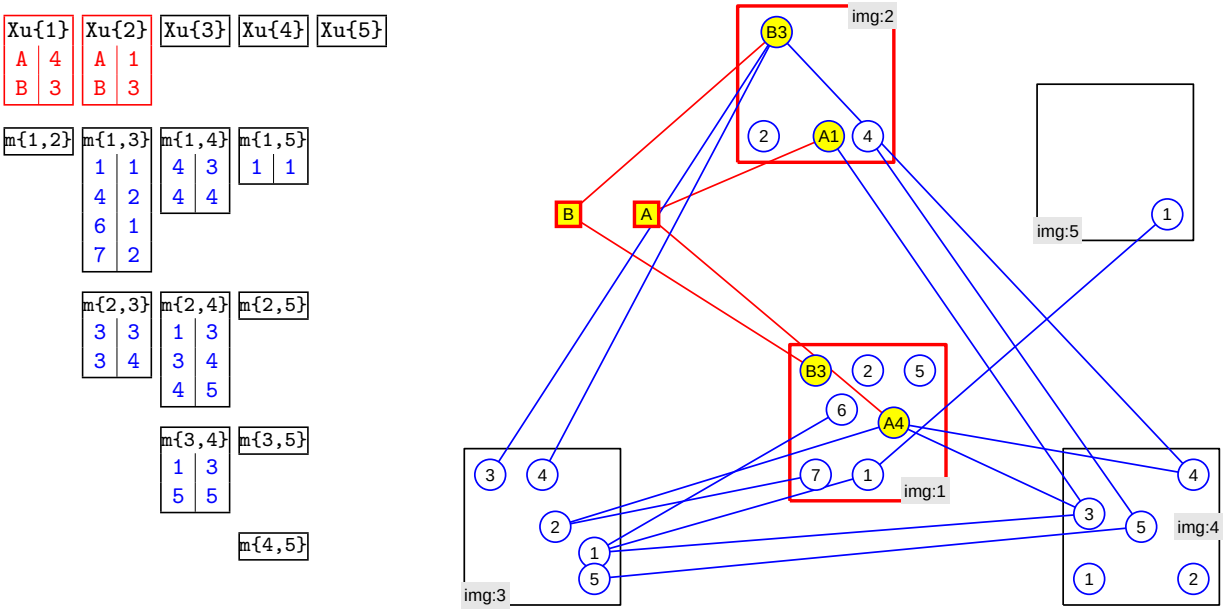

Fig. 3: The camera cluster and the point cloud initialised. The cameras belonging to the cluster are marked red, the scene-to-image correspondences that are confirmed are red as well.

2.2.3 Propagation of Tentative Scene-to-Image Correspondences (1)

The newly confirmed scene-to-image correspondences are now propagated along tentative image-to-image correspondences into tentative scene-to-image correspondences. The propagation is done into cameras in the neighbourhood of the cluster. (Not inside, since there are no image-to-image correspondences inside the cluster). The image-to-image correspondences used for propagation are removed. It is clear, that after the propagation, every interest point related by at least one confirmed scene-to-image correspondence is no more related by any image-to-image correspondence.

This propagation is done automatically by the `corresp_start` function.

The situation is depicted on the following three figures. The first two (Fig. 4, Fig. 5) show the propagation from the first and the second camera, respectively. The final situation is shown on the third figure (Fig. 6).

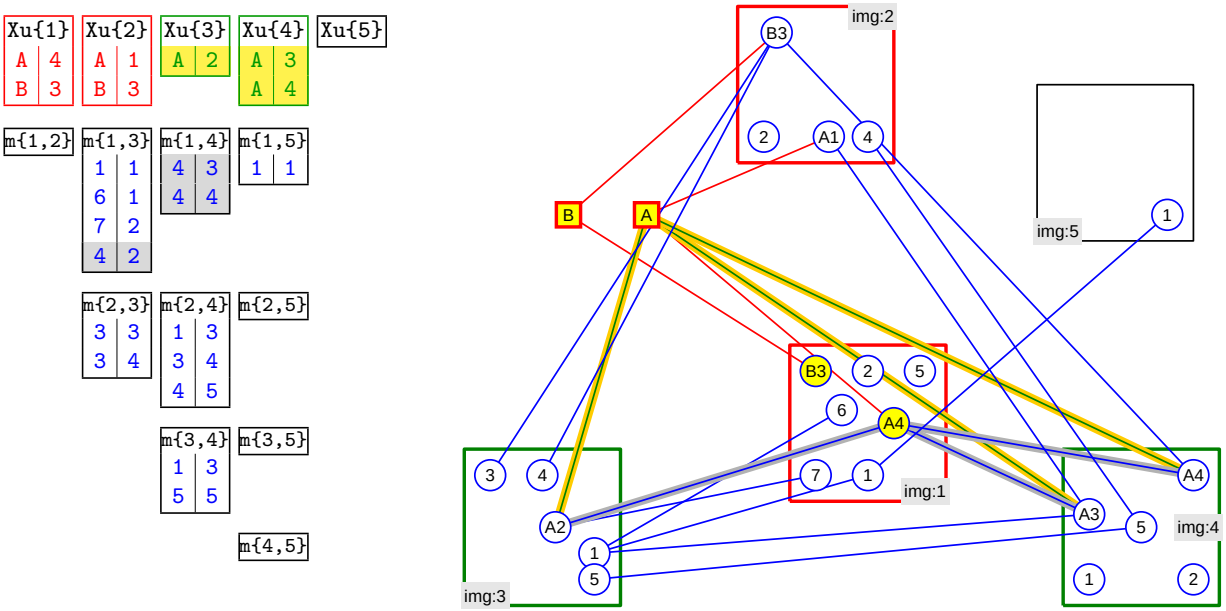

Fig. 4: Correspondence propagation from the first camera. The used and later removed correspondences are emphasised in gray, while the new correspondences are emphasised in yellow.

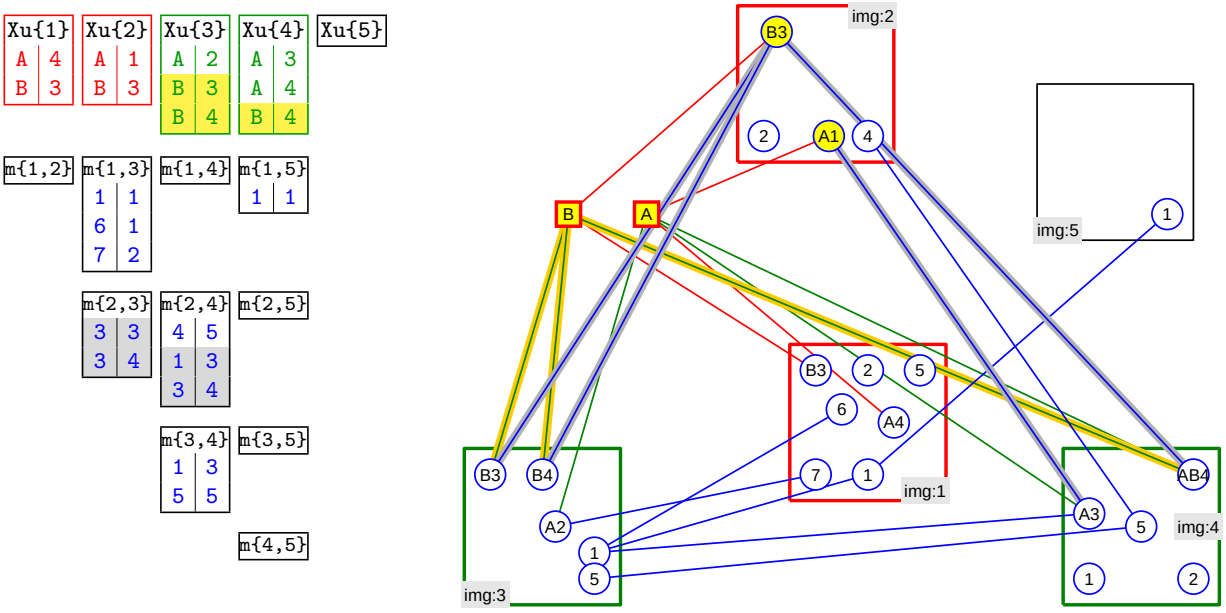

Fig. 5: Correspondence propagation from the first camera. The same colours as in the previous figure.

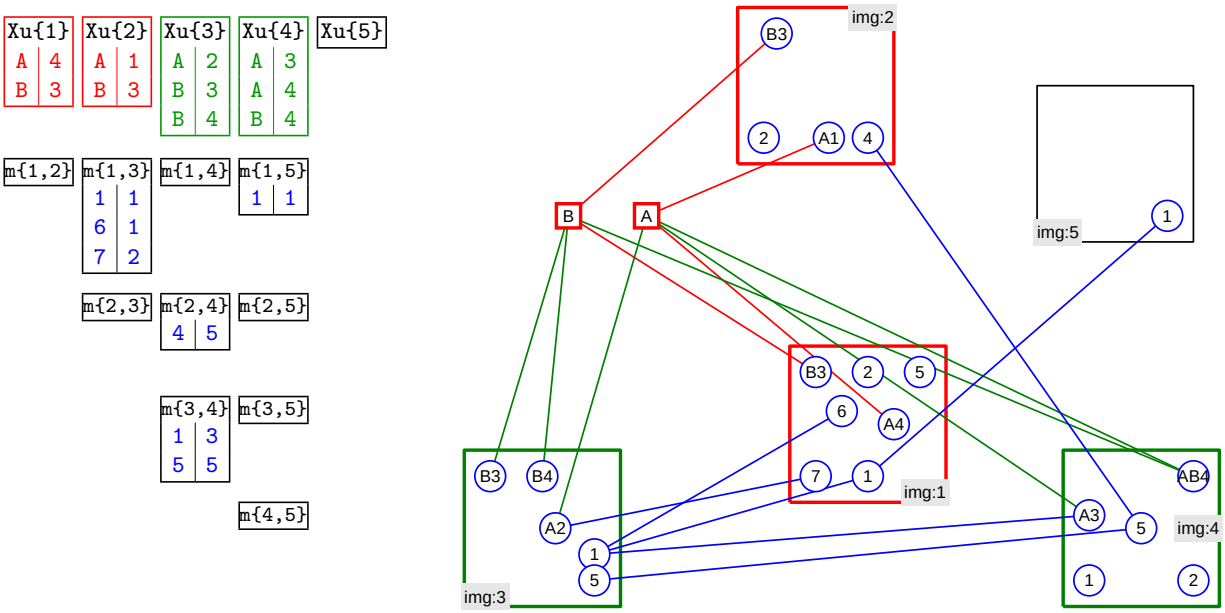

Fig. 6: Correspondences propagated. Tentative scene-to-image correspondences are marked green. Recall that confirmed scene-to-image correspondences are marked red.

2.2.4 Attaching New Camera into the Cluster

A camera, not already in the cluster, but with a sufficient number of tentative scene-to-image correspondences, is selected. The camera is reconstructed (e.g. RANSAC with P3P) from these correspondences. Scene-to-image correspondence inliers are confirmed, outliers are removed.

```
% list of cameras with tentative scene-to-image correspondences
ig = corresp_get_green_cameras( corresp );
% counts of tentative correspondences in each 'green' camera
Xucount = corresp_get_Xucount( corresp, ig );
i = 3;  % select the camera to be attached, based e.g. on the Xucount
Xu = corresp_get_Xu( corresp, i );  % get scene-to-image correspondences

Obtain the i-th camera pose (R, t) and a set of inliers using the scene-to-image correspondences in Xu.

xinl = [1 2];  % obtained inliers – indices to Xu{i}
corresp = corresp_join_camera( corresp, i, xinl );
```

This step is depicted in the next two figures. The first one (Fig. 7) emphasises inliers and outliers, the second one (Fig. 8) shows the final state. Note that inliers in `Xu{3}` become confirmed (red).

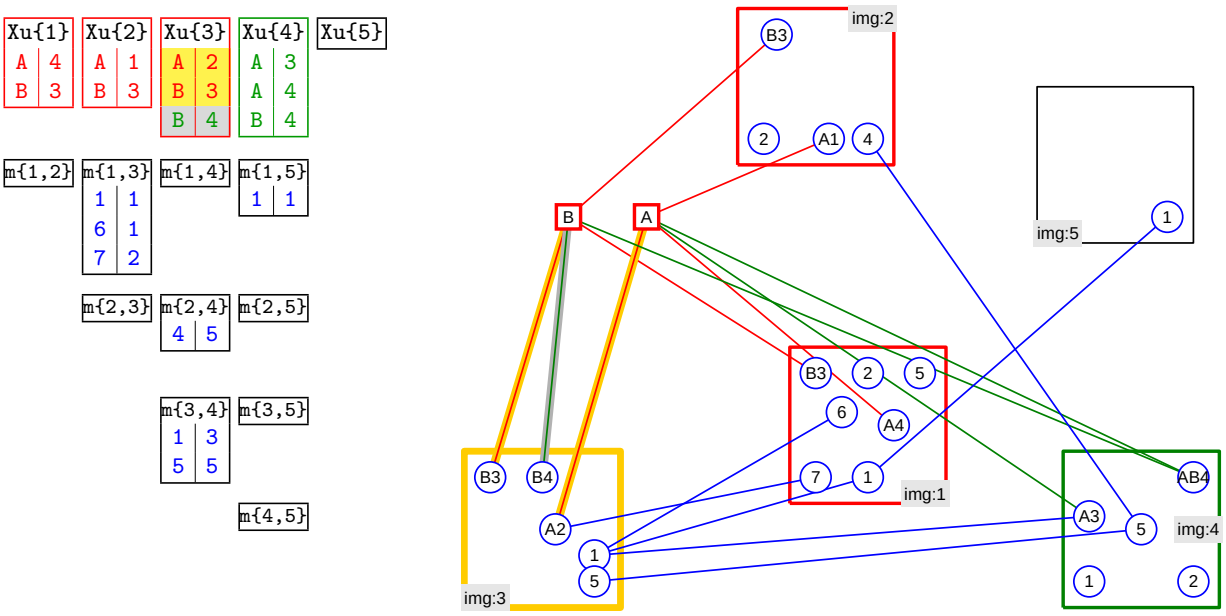

Fig. 7: Attaching a new camera. Scene-to-image correspondence inliers and outliers are emphasised yellow and gray, respectively.

2.2.5 Propagation of Tentative Scene-to-Image Correspondences (2)

Again, the newly confirmed scene-to-image correspondences in the attached camera are now propagated into tentative scene-to-image correspondences in the other cameras, by the same algorithm as in the previous propagation. The propagation can lead outside as well as inside the cluster, in this case.

This step is done automatically by the `corresp_join_camera` function.

The step is again shown in the following two figures. The first one (Fig. 9 shows the propagation, emphasising new and used-and-removed correspondences. The final situation is shown in the second figure (Fig. 10). Note that tentative scene-to-image correspondence (green) emerge in the camera 1 inside the cluster.

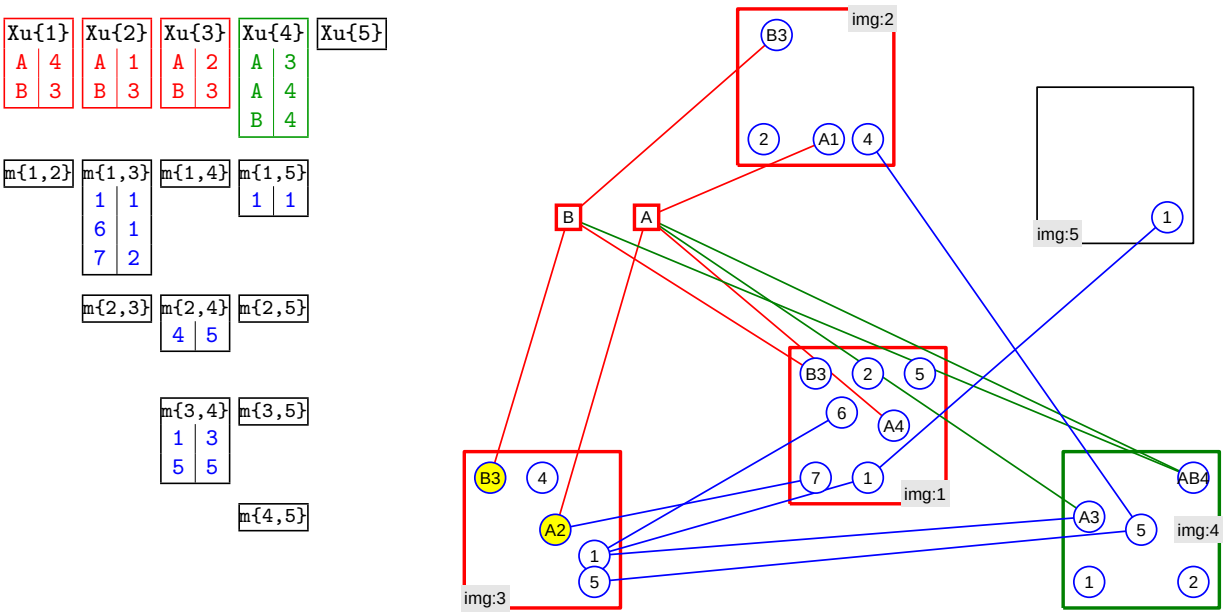

Fig. 8: New camera attached.

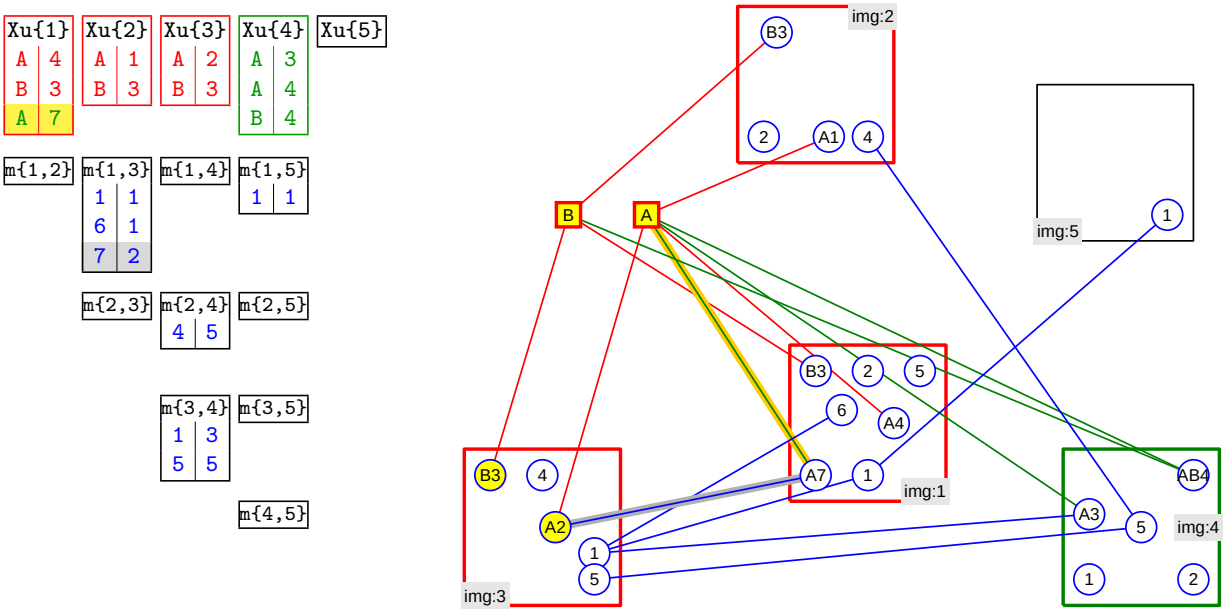

Fig. 9: Correspondence propagation from the attached camera. Used and removed correspondences are emphasised gray, new correspondences are emphasised yellow.

2.2.6 Reconstructing New Points

Remaining image-to-image correspondences between the attached camera and the other cameras in the cluster are verified, inliers are used to create new scene points and confirmed scene-to-image correspondences. Then all these image-to-image correspondences are removed.

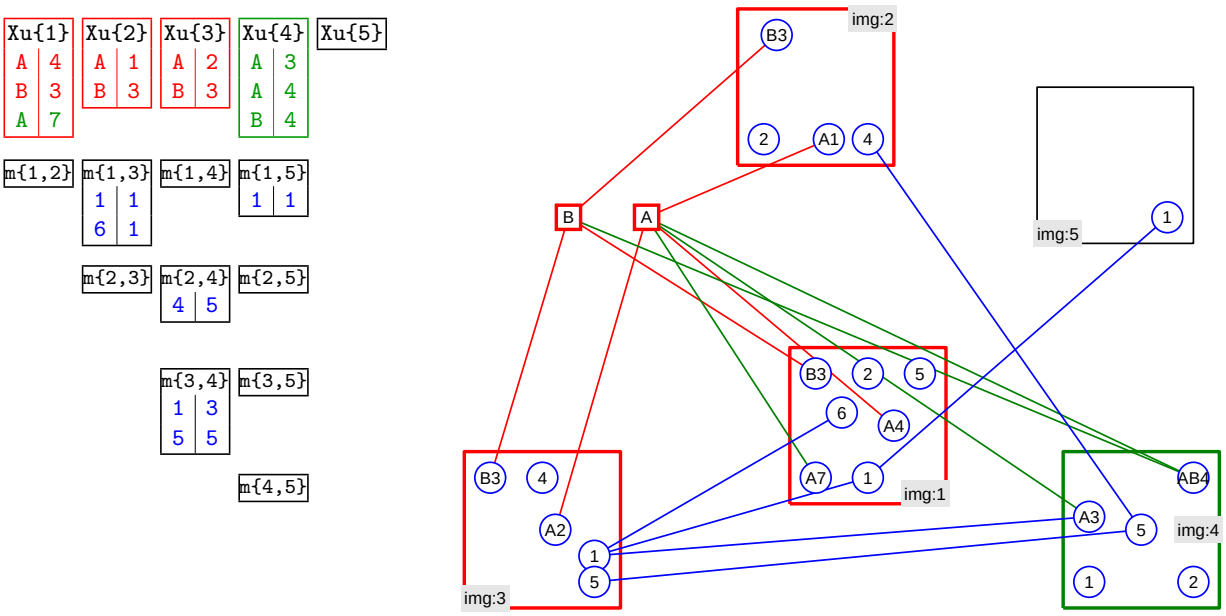

Fig. 10: Correspondences propagated from the attached camera (red for confirmed, green for tentative).

```
% i = 3; already set
ilist = corresp_get_cneighbours( corresp, i ); % List of cameras in the cluster that are related to the attached camera by some image-to-image correspondences.

ic = 1; % a camera in the neighbourhood (must be iterated through ilist).
m = corresp_get_m( corresp, i, ic ); % get remaining image-to-image correspondences

Reconstruct new scene points using the cameras i and ic and image-to-image correspondences m. Sets of inliers and new scene points' IDs are obtained.

inl = [1]; % Obtained inliers–indices to m
xid = ['C']; % scene points' IDs
corresp = corresp_new_x( corresp, i, ic, inl, xid );
```

The step is demonstrated on the following two images. The first one (Fig. 11) emphasises the inliers and new scene-to-image correspondences, the second one (Fig. 12) shows the final state.

2.2.7 Propagation of Tentative Scene-to-Image Correspondences (3)

Again, the new confirmed scene-to-image correspondences are now propagated, done automatically by the `corresp_new_x` function. See Fig. 13, Fig. 14, and Fig. 15.

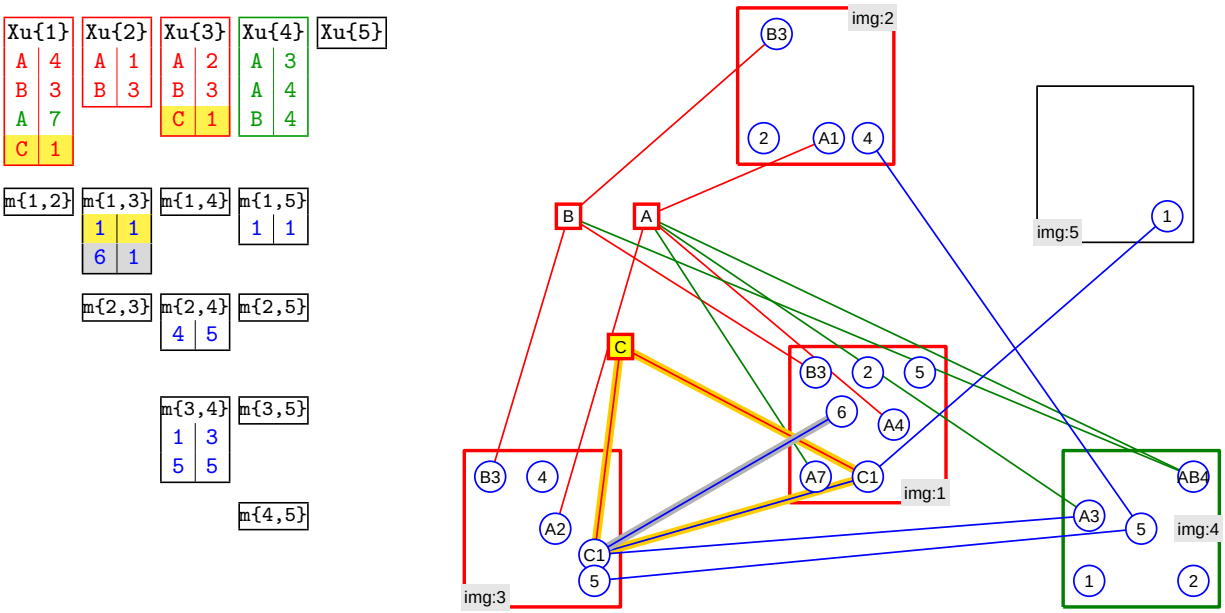

Fig. 11: Reconstructing new points from inlier image-to-image correspondences (emphasised yellow) between the attached camera and the cluster. The new (confirmed) scene-to-image correspondences are emphasised yellow.

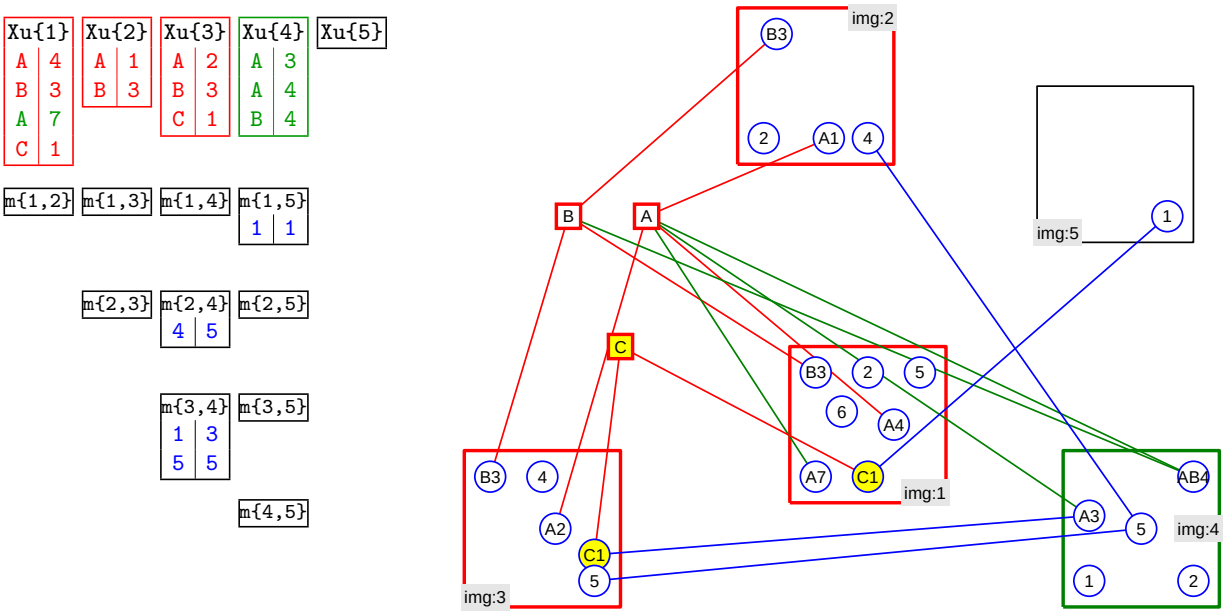

Fig. 12: New points reconstructed.

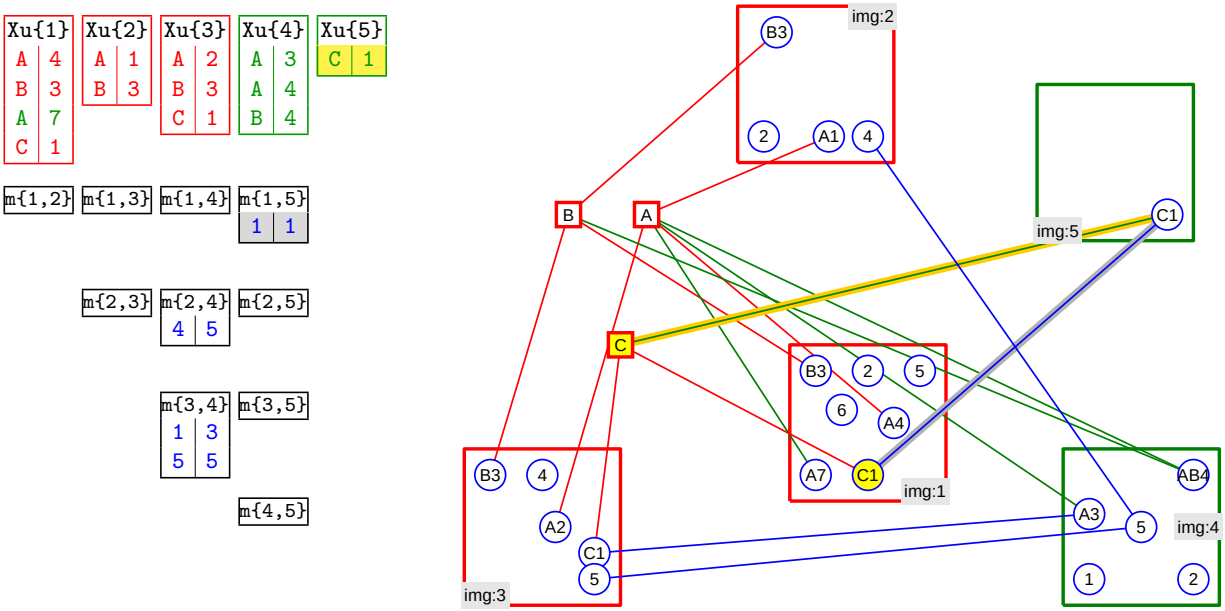

Fig. 13: Propagation of new confirmed scene-to-image correspondences.

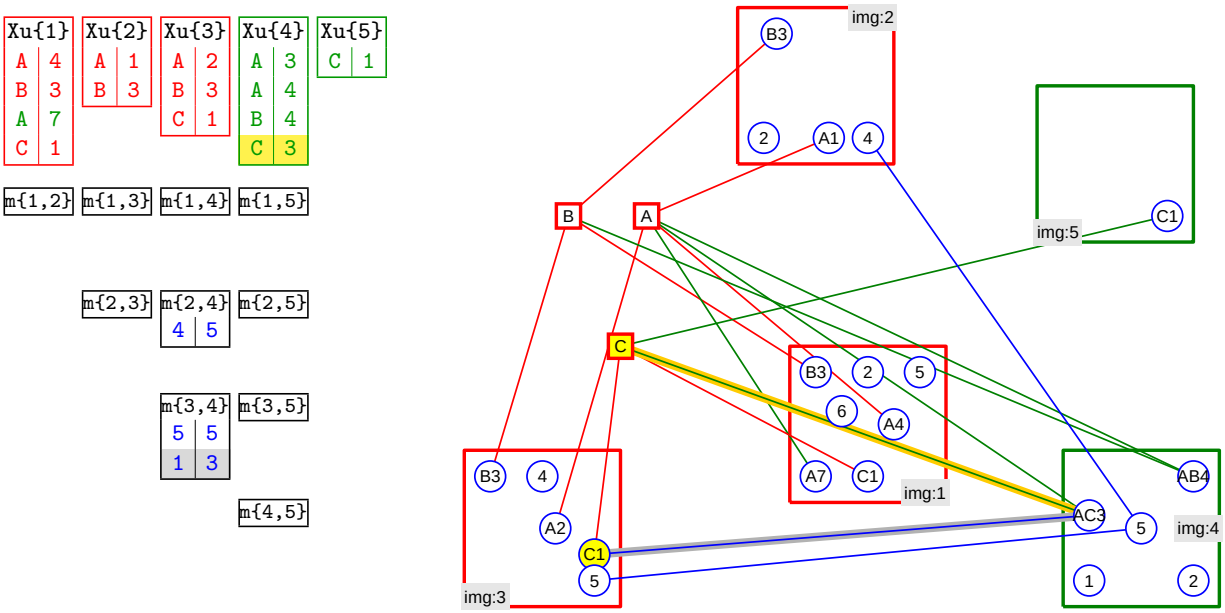

Fig. 14: Propagation of new confirmed scene-to-image correspondences.

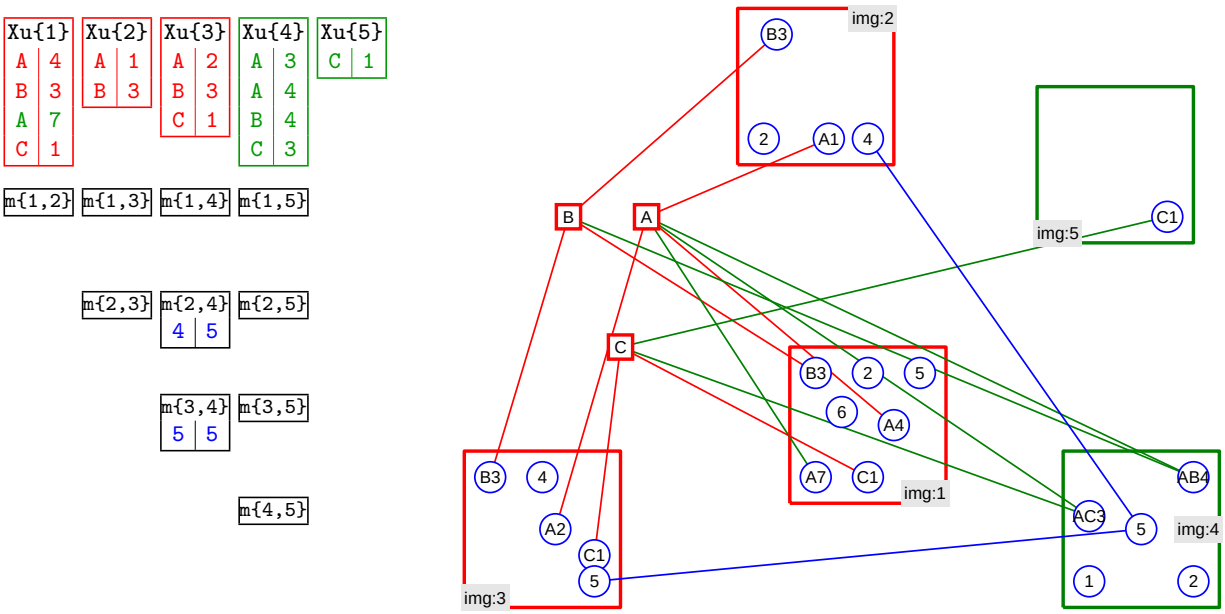

Fig. 15: New confirmed scene-to-image correspondences propagated.

2.2.8 Verification of Tentative Scene-to-Image Correspondences in the Cluster.

During the previous propagations, some tentative scene-to-image correspondences can emerge also in the cameras already selected in the cluster. These must be geometrically verified. The good ones are turned into the confirmed state, the others are removed. Fig. 16.

```
ilist = corresp_get_selected_cameras( corresp ); % list of all cameras in the cluster

ic = 1; % a camera in the cluster (must be iterated through ilist).
[Xu Xu_verified] = corresp_get_Xu( corresp, ic );
Xu_tentative = find( ~Xu_verified );

Verify (by reprojection error) scene-to-image correspondences in Xu_tentative. A subset of good
points is obtained.

corr_ok = []; % The subset of good points—there is no one here.
corresp = corresp_verify_x( corresp, ic, corr_ok );
```

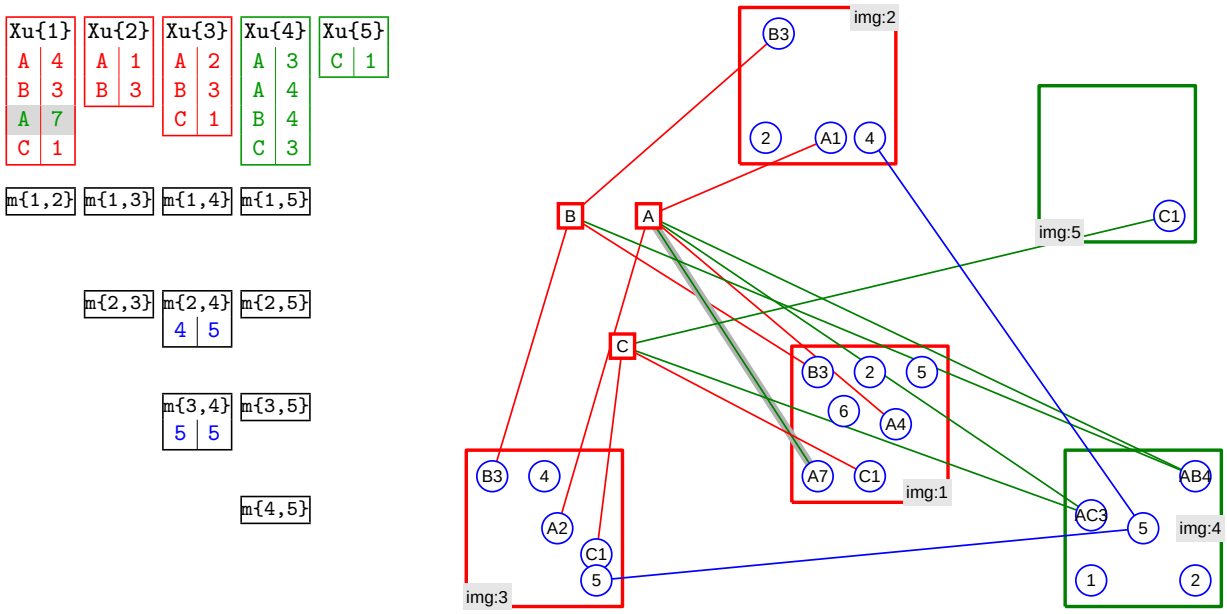

Fig. 16: Verification in the cluster. Tentative scene-to-image correspondences between the scene and the cluster are removed (emphasised gray).

2.2.9 Propagation of Tentative scene-to-image Correspondences (4)

The newly verified scene-to-image correspondences must be propagated. Note that since there are no image-to-image correspondences inside the camera cluster, the propagation affects only cameras outside the set. Done automatically by the `corresp_verify_x` function.

There are none in the example now.

2.2.10 Finalize New Camera

After all new points are reconstructed and verification is done, the new camera is finalised. This step mainly checks for data consistency. At this moment, there must be no **tentative** scene-to-image correspondences leading to the cluster and no image-to-image correspondences inside the cluster.

```
corresp = corresp_finalize_camera( corresp );
```

2.2.11 Repeat Attaching the Camera

Repeat Step 4 while there exists a camera outside the cluster with sufficient number of tentative scene-to-image correspondences.

```
corresp = corresp_join_camera( corresp, 4, [3] );
```

The attaching is shown in Fig. 17. No effect of propagation is in the example here.

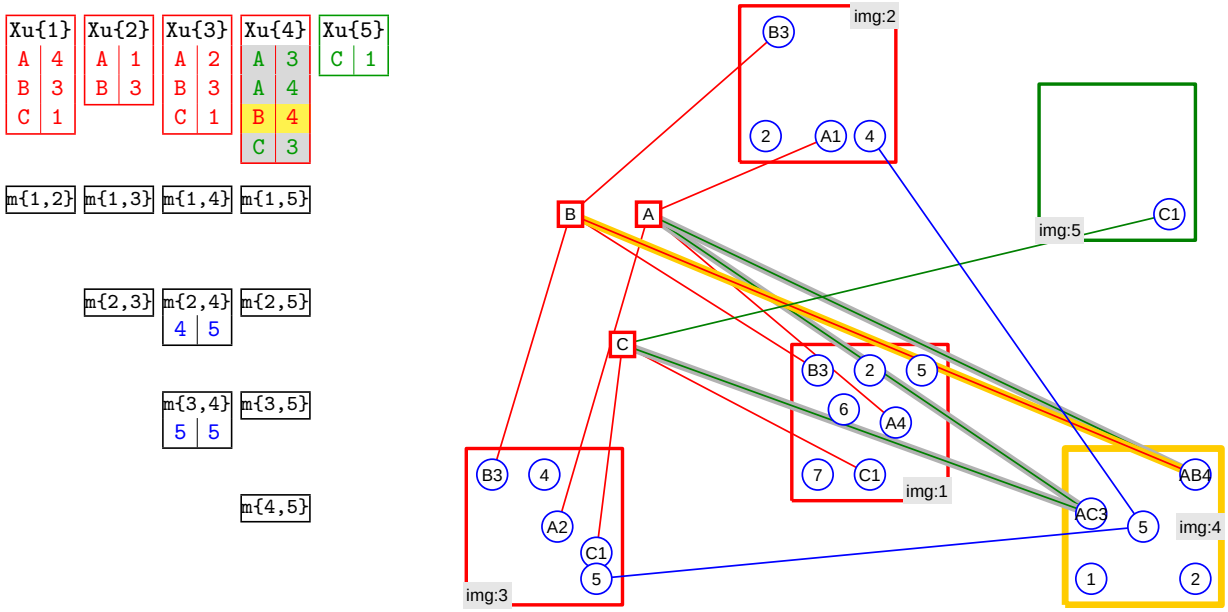

Fig. 17: Attaching the next camera.

After attaching the camera, new points are created (Fig. 18), and propagated (Fig. 19).

```
corresp = corresp_new_x( corresp, 3, 4, [1], [ 'D' ] );
```

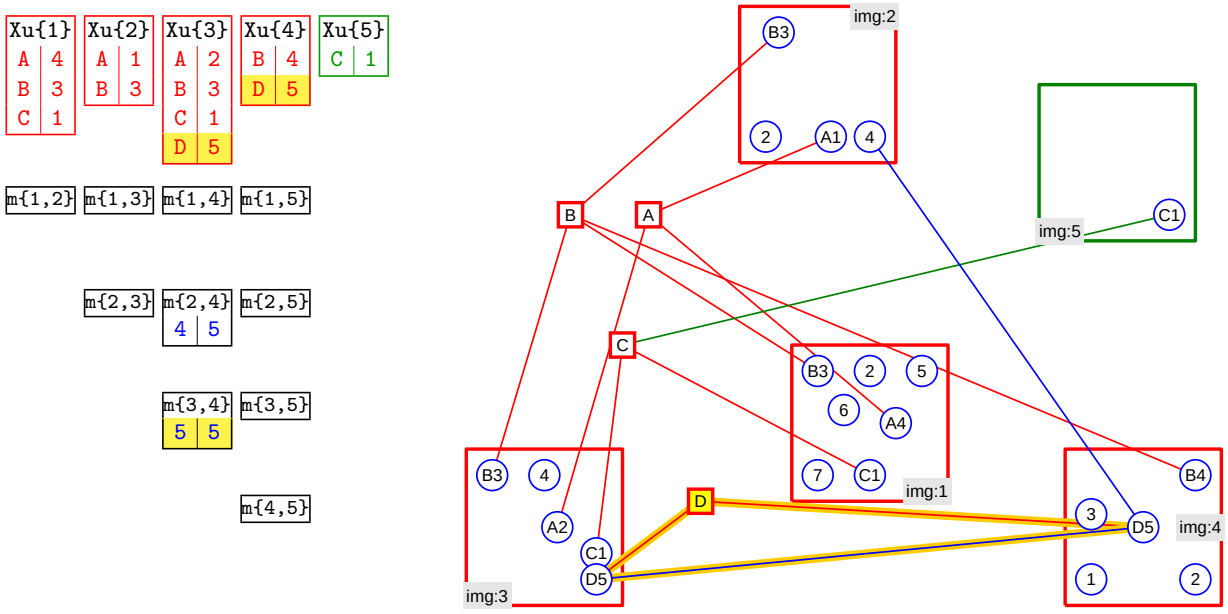

Fig. 18: Reconstruction of new points.

Some correspondences propagate into the cluster and are verified and turned into the confirmed state (Fig. 20).

```
corresp = corresp_verify_x( corresp, 2, [3] );
```

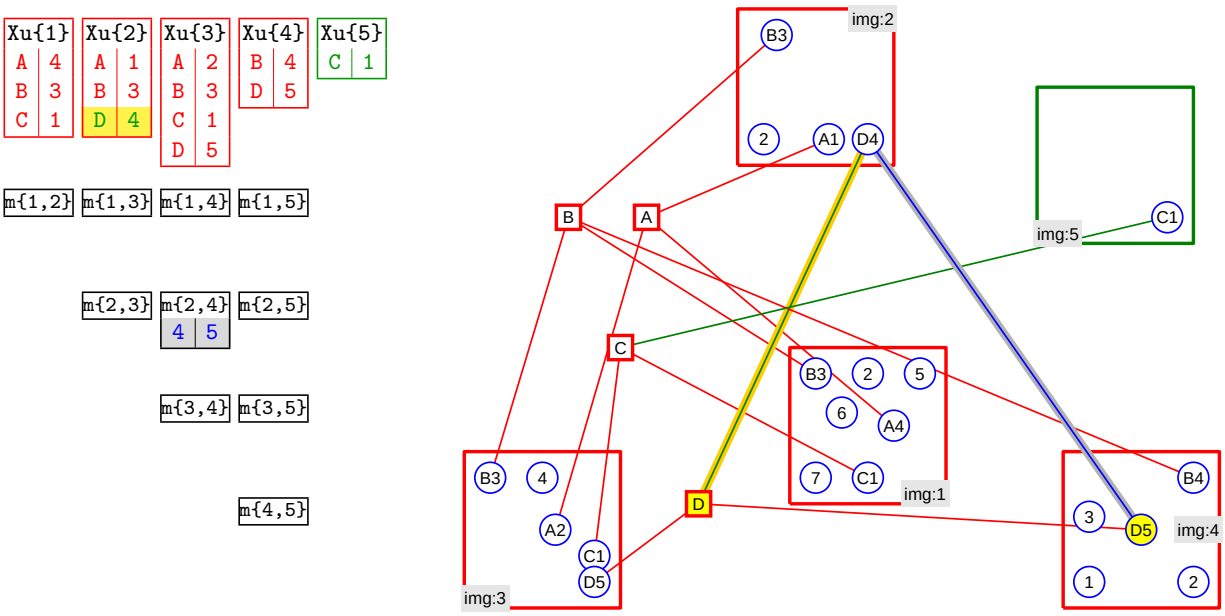

Fig. 19: Propagation of new scene-to-image correspondences.

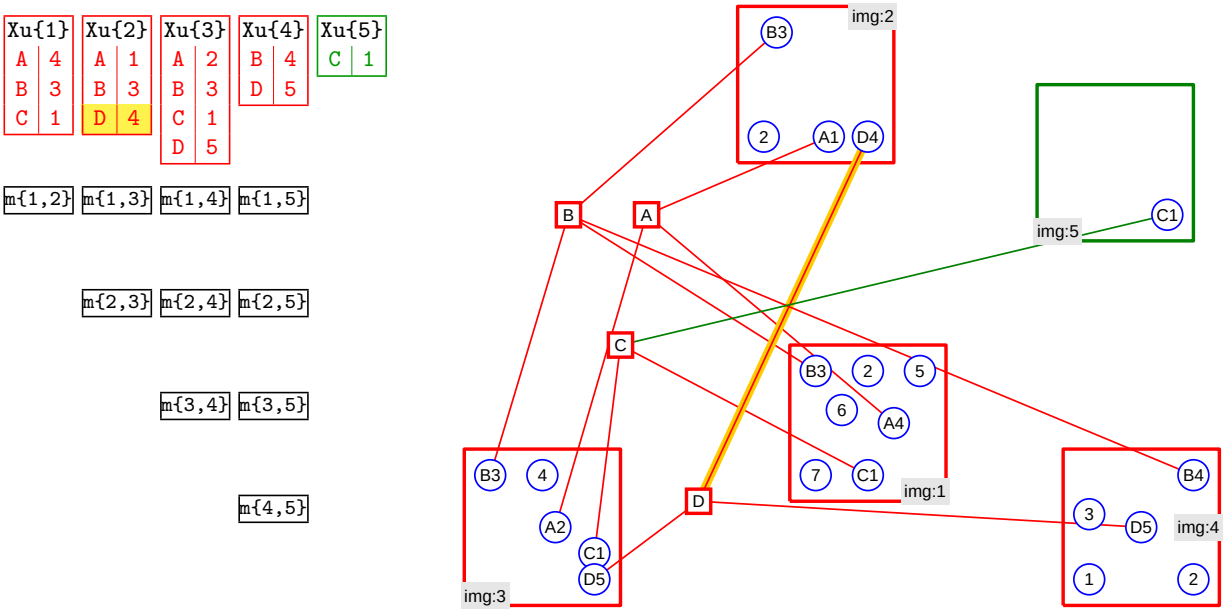

Fig. 20: Verification of tentative scene-to-image correspondences in the cluster.

Attaching of the last camera is now finalised.

```
corresp = corresp_finalize_camera( corresp );
```

Let us assume, that it is not possible to attach camera 5. So the final state is in Fig. 21

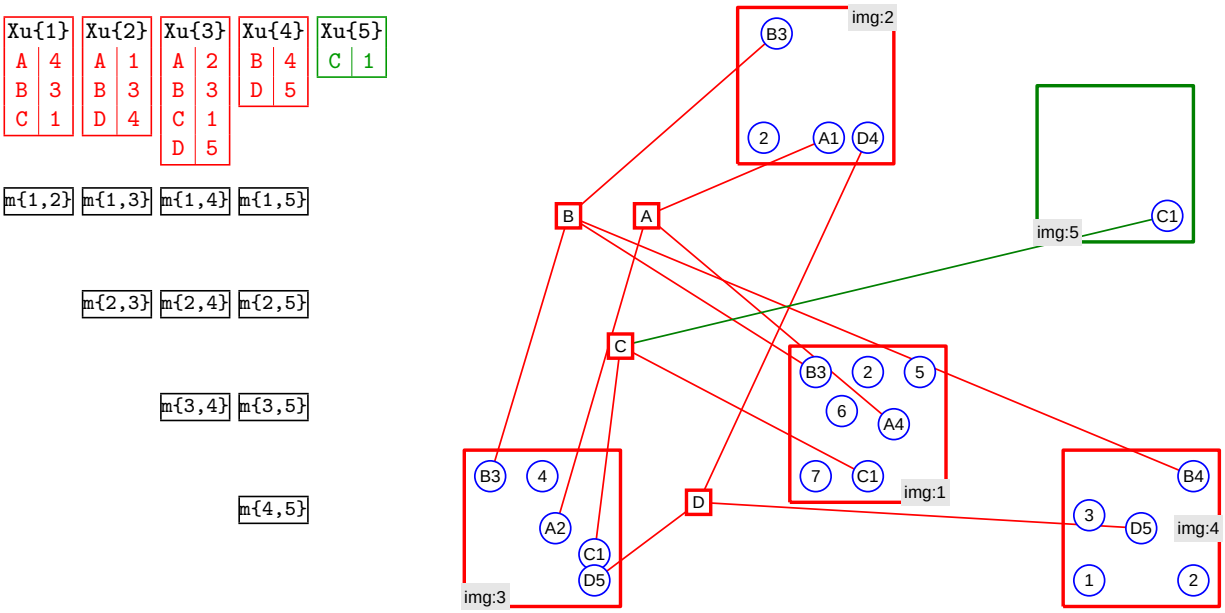

Fig. 21: Final state of correspondence manipulation. There is a camera cluster and a point cloud, and these two are related with confirmed scene-to-image correspondences.